On the Use of API Calls for Detecting Repackaged Malware Apps: Challenges and Ideas

Kobra Khanmohammadi
Department of Electrical and Computer Engineering
Concordia University
Montréal, QC, Canada
k_khanm@encs.concordia.ca

Raphaël Khoury
Dep. of Computer Science and Mathematics
Université du Québec à Chicoutimi
Chicoutimi, QC, Canada
raphael.khoury@uqac.ca

Abdelwahab Hamou-Lhadj
Department of Electrical and Computer Engineering
Concordia University
Montréal, QC, Canada
abdelw@ece.concordia.ca

***Abstract*—Traces of API calls from mobile applications are a very valuable source of information for multiples security analyses including the detection of malware and repackaged apps. Although API call traces are widely, extracting and using API call traces for the detection of repackaged apps remains a challenging task. In this paper, we briefly review (a) the challenges associated with using API calls and (b) the limitations of malware detection approaches that rely on API calls.**

***Keywords*—API Call Traces, Mobile Application Reliability and Security, Malware Detection.**

I. Introduction

A common practice in security analysis of Android mobile applications (apps) is to extract app's attributes such as opcodes, API calls, images, resources, or user interface graphs and study them to understand how the app behaves with the objective of detecting malware and other undesirable behaviors. Among these features, API call traces have shown to be the most reliable, given that it is often difficult for an attacker to manipulate them [1].

Generating API call traces from an Android app is a challenging problem. In a normal situation, a simple static analysis of the app source code would suffice, but, obfuscation makes static analysis almost impossible [3, 4]. Dynamic analysis can be used to overcome obfuscation. This technique consists in executing the app with various inputs and recording the trace of API calls. Approaches based on dynamic analysis (e.g., [5][16]), however, cannot guarantee full code coverage due to incomplete input. In addition, there exist malware families that hinder dynamic analysis by using techniques such as emulator detection or by adding logical bombs, triggered upon the satisfaction of specific time conditions [15]. Limited work has been carried out using hybrid approaches [6].

The main goal of this abstract is to discuss the challenges of using API call traces for detecting repackaged Android apps.

II. Difficulties in Generating API Call Traces

Many obstacles hinder the generation of API call traces from Android apps including reflection call, dynamic loading, and emulator detection.

A. Reflection Calls

Reflection is a functionality that allows developers to identify a method for calling at run-time. It is used in legitimate apps mostly for the purpose of implementing generic functionalities, maintaining backward compatibility, hiding the main functionalities, or accessing the internal APIs reserved only for system apps [7]. The use of reflection calls in malware makes the extraction of API call traces challenging [4]. In a recent study, Li et al. [7] proposed a static analysis method to identify the methods called by reflection calls in an app. It searches an app for the most common patterns of reflection calls and identifies the reflective method names. Reflection calls remain a barrier to code analysis if the name of the called method is encrypted.

B. Dynamic Loading

Another functionality available to Android app developers is the use of the package DexClassLoader[1] to load a library at run-time and then call those library methods through reflection calls at run-time. The loaded library may not be present in the apk file and is loaded from an external source in the network. Dynamic analysis approaches are largely able to load those parts of the code while the app is executing and study them. The evaluation of these approaches would require a dataset that contains malware samples for which the target network location is still available. By studying the AndroZoo dataset [8], we found that in most of the samples that use dynamic loading, the network address is not available any more. Providing adequate samples in datasets such as DroidBench[2] would be helpful.

C. Emulator Detection

Some malware samples use emulator detection systems to evade monitoring of code using an emulator [15]. Emulator detection systems examine a selected set of features of the underlying device at run-time to ensure that the app is running in a real device and adapt the execution accordingly. Researchers need to simulate the real device's state in a virtual environment to combat such malware. An alternative solution, suggested by some researchers, is to identify those parts of the code that perform emulator detection [15].

III. Limitations of Existing Malware Detection Approaches Using API calls

A. Curse of using machine learning approaches

Machine learning algorithms are widely used to differentiate malware and benign apps. The algorithms learn the distinctive behavior of malicious operations through the set of features extracted from benign and malware apps. The main advantage of using a machine learning approach is that the detection algorithm does not require predefined malware signatures. API call frequency, API call graphs, and API call

[1]https://developer.android.com/reference/dalvik/system/package-summary

[2] https://github.com/secure-software-engineering/DroidBench

sequences have all been used extensively as a feature set for machine learning algorithms (e.g., [1]). The disadvantage of using machine learning is a high risk of false negatives. The challenge of how to set the proper thresholds between malware or benign apps remains also an open question. Another issue relates to the fact that even after an app is found to be a malware it is not clear what malicious operations the app contains. We need machine learning based solutions that do not only flag malicious behavior but also provide guidance to security analysts on how to identify the underlying causes.

B. *Evaluation of malware detection approaches*

It is a common practice to evaluate a malware detection approach using a dataset of benign and malware apps. Several different datasets have been employed for this purpose. Comparing the results of techniques tested on different datasets is not straightforward. AndroZoo [8], a large dataset of benign and malware apps, has been publicly available since 2015, and has been used by multiple researchers. It would be good to have a dataset of API calls that characterise the malicious code embedded in AndroZoo malicious apps. Such dataset would not only help researchers compare their results consistently, but also save them time analyzing malware apps to find out the associated API calls.

C. *Adware detection*

In our empirical study on repackaged apps [2], we showed that a large number of malware apps are adware. We also showed that they exhibit a pattern of API calls that is very similar to those of the original benign apps, and furthermore that they employ the same ad libraries as benign apps. Because of these similarities, detecting adware using API calls is more difficult than detecting other categories of malware. Studying adware and proposing detection approaches targeted specifically to this category of malware is an important goal of future research since the presence of adware does not only affect users and developers, but also ad companies.

D. *Protectinng apps against cloning*

Several studies propose ways to protect apps against repackaging by checking whether the code has changed during execution (e.g., [11] [12]). The common approach is to record features such as API calls of each app and check them while the app is executing to detect tampering. Other studies (e.g., [13]) use watermarking and defer the responsibility for detecting malicious behavior to end users. the end user to detect if the original app is manipulated. These approaches may not be practical because they either require changes to the DVM or rely on end users with limited knowledge in this area. Since most malware embedded in repackaged apps is adware as we showed in [2], we propose to work towards approaches that involve ad companies in the authentication of apps. Ad companies can provide an SDK for developers to query advertisements. They can also record the developers' identity for payment purposes. Additional studies are needed to assess the applicability of authenticating apps by ad companies.

E. *Difficulties in using app similarity*

Many studies (e.g., [14]) use API call traces to detect repackaged apps. These studies assume that a repackaged app keeps the same appearance as the original one in order to offer the same experience to users. Besides the computational overhead of these approaches, the main constraint is that they require pair-wise comparison between two apps to detect a repackaged app. A practical application of this approach would require a large repository of apps, which may cause scalability problems if API calls are the main features used for malware detection. This is because of the high computational time associated with generating and managing large API call traces. We need to work towards ways to reduce the size of API traces using trace abstraction, filtering and simplification techniques such as the ones that have extensively studied in traditional software systems (see [18] for examples).

References

[1] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. E. Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid : Detecting Android Malware by Building Markov Chains of Behavioral Models," *ACM Transactions on Privacy and Security, vol. 22, no. 2,* 2019.

[2] K. Khanmohammadi, N. Ebrahimi, A. Hamou-Lhadj, R. Khoury, "Empirical Study of Android Repackaged Apps," *Springer Journal on Empirical Software Engineering,* pp 1-43, 2019.

[3] Q. Guan, H. Huang, W. Luo, and S. Zhu, "Semantics-based repackaging detection for mobile apps," in *Proc. of the Symposium on Engineering Secure Software and Systems,* pp 89-105. 2016.

[4] M. Hammad, J. Garcia, and S. Malek, "A Large-Scale Empirical Study on the Effects of Code Obfuscations on Android Apps and Anti-Malware Products," in *Proc. of the 40th International Conference on Software Engineering,* pp 421–431, 2018.

[5] A. Aldini, F. Martinelli, A. Saracino, and D. Sgandurra, "Detection of repackaged mobile applications through a collaborative approach," *Wiley Journal on Concurrency and Computation: Practice & Experience*, vol.27, no.11, pp 2818–2838, 2015.

[6] M. Y. Wong and D. Lie, "IntelliDroid : A Targeted Input Generator for the Dynamic Analysis of Android Malware," in *Proc. of the Network and Distributed System Security Symposium,* vol. 16, 2016.

[7] L. Li, T. F. Bissyandé, D. Octeau, and J. Klein, "DroidRA : Taming Reflection to Support Whole-Program Analysis of Android Apps," in *Proc. of the 25th International Symposium on Software Testing and Analysis,* pp 318–329, 2016.

[8] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: Collecting Millions of Android Apps for the Research Community," in *Proc. of Mining Software Repositories Conf.*, pp. 468–471, 2016.

[9] L. Luo, Y. Fu, D. Wu, S. Zhu, and P. Liu, "Repackage-proofing Android Apps," in *Proc. of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks,* pp 550–561, 2016.

[10] Q. Zeng, L. Luo, Z. Qian, X. Du, and Z. Li, "Resilient Decentralized Android Application Repackaging Detection Using Logic Bombs," in *Proc. of the International Symposium on Code Generation and Optimization*, pp 50–61, 2018.

[11] W. Zhou, X. Zhang, and X. Jiang, "AppInk: watermarking android apps for repackaging deterrence," in *Proc. of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security,* pp 1-12, 2013.

[12] C. Ren, K. Chen, and P. Liu, "Droidmarking: Resilient software watermarking for impeding android application repackaging," in *Proc. of the 29th ACM/IEEE International Conference on Automated Software Engineering,* pp 635-646, 2014.

[13] Q. Zeng, L. Luo, Z. Qian, X. Du, and Z. Li, "Resilient Decentralized Android Application Repackaging Detection Using Logic Bombs," in *Proc. of the International Symposium on Code Generation and Optimization,* pp. 50–61, 2018.

[14] S. Jiao, Y. Cheng, L. Ying, P. Su, and D. Feng, "A rapid and scalable method for android application repackaging detection," in *Proc. of the International Conference on Information Security Practice and Experience,* pp 349-364. 2015.

[15] X. Pan, X. Wang, Y. Duan, X. Wang, and H. Yin, "Dark Hazard: Learning-based, Large-Scale Discovery of Hidden Sensitive Operations in Android Apps," in *Proc. of the Network and Distributed System Security Symposium,* 2017.

[16] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android," in *Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, pp 15–26, 2011.

[17] K. Khanmohammadi, A. Hamou-Lhadj, "HyDroid: A Hybrid Approach for Generating API Call Traces from Obfuscated Android Applications for Mobile Security," in *Proc. of the International Conference on Software Quality, Reliability and Security (QRS),* pp 168-175, 2017.

[18] A. Hamou-Lhadj, "Techniques to simplify the analysis of execution traces for program comprehension," Ph.D. Dissertaiton, University of Ottawa, 2015.